R E P O R T R E S U M E S

ED 011 647 AL 000 058

A PROGRAM FOR TRANSFORMATIONAL SYNTACTIC ANALYSIS.

BY- PETRICK,S.R.

PUB DATE 66

EDRS PRICE MF-$0.09 HC-$1.96 49P.

DESCRIPTORS- *COMPUTATIONAL LINGUISTICS, *SYNTAX, *TRANSFORMATION THEORY (LANGUAGE), *ENGLISH, SENTENCE ANALYSIS, COMPUTER PROGRAMS, LISP PROGRAMMING LANGUAGES

A CLASS OF TRANSFORMATION GRAMMARS IS DEFINED AND A COMPUTER PROGRAM FOR SENTENCE ANALYSIS IS DESCRIBED AND DOCUMENTED WITH RESPECT TO THIS CLASS. THE PROGRAM EXISTS IN PURE LISP FORM AND IN MIXED LISP AND IBM 7090 ASSEMBLY LANGUAGE FORM. THE PAPER CONTAINS INFORMATION TO PERMIT THE USER TO WRITE HIS OWN TRANSFORMATIONAL GRAMMAR. COMPUTER PROGRAM OPERATING DETAILS AND FORMATS ARE INCLUDED. THE PRINCIPAL DIFFERENCE BETWEEN THIS WORK ON SYNTACTIC ANALYSIS AND THAT OF OTHER GROUPS IS THAT FEW OTHER EFFORTS HAVE BEEN DIRECTED TOWARD THE ANALYSIS OF SENTENCES IN TERMS OF TRANSFORMATIONAL GRAMMAR. (KL)

ED011647

A Program for Transformational Syntactic Analysis

By S. R. Petrick

Introduction

We have been concerned with the problem of syntactic analysis with respect to a transformational grammar for more than three years now, and during that time we have developed a sequence of computer programs that implement our ideas on transformational analysis. These programs for the most part have been written in the LISP programming language and debugged on the AFCRL Univac M-460 LISP System. As might be expected the basic program has undergone a continual sequence of changes. These modifications stem from several causes among which are the following:

(1) Analysis of sentences with respect to specific transformational grammars has uncovered several fundamental errors in our analysis procedure. This caused us to modify our analysis algorithm and, of course, to make corresponding changes to the LISP implementation of that algorithm.

(2) Changes have been made in the underlying class of transformational grammars for which the analysis procedure is valid. These changes were made to permit the consideration of grammars that reflect current developments in transformational theory.

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

(3) Actual user experience has dictated certain changes in input and output formats. Several output options have been added to aid in the debugging of transformational grammars.

(4) Excessive time requirements encountered in the analysis of specific sentences have suggested changes which have increased the efficiency of the analysis program.

Changes of the last three types are still being made. In spite of this it seems appropriate at this time to document and make generally available a version of the program which has been tested on the IBM 7044 and 7094 LISP Systems as well as on the Univac M-460 LISP System. It is hoped that this program will be of use to transformational grammar writers who wish to verify that their grammars assign structural descriptions as intended. It is further hoped that the availability of this program may encourage people who are interested in working toward the realization of a natural language computer input capability to consider the problem of mapping structural descriptions assigned to a sentence by a transformational grammar into appropriate computer code.

In this paper we will precisely define a class of transformational grammars and we will describe and document a computer program for the analysis of sentences with respect

to grammars of this class. This program is at present unique in constituting an analysis procedure that is valid for a non-trivial class of transformational grammars. We will not present the linguistic motivation for our formalization of transformational grammars, nor will we discuss in detail the analysis algorithm upon which the computer program is based. These topics are discussed elsewhere [1,2]. We will comment briefly on the relationship of our approach to that of others, but this comparison will be deferred until we have discussed our own work.

Class of Grammars Considered

The syntactic component of a transformational grammar, as we define it, consists of two subcomponents, a base component and a transformational component. The base component specifies a set of trees or labelled brackettings that we will call "base trees" or "deep structures". The transformational component maps certain base trees into other trees that we will refer to as "surface trees". These surface trees are, of course, also labelled brackettings, and their debracketizations constitute the sentences of the language specified by the syntactic component of a transformatinnal grammar.

The base component will be taken to be a context free (CF) grammar.* We will not formalize a CF grammar here but will merely describe those conventions we have fixed for our convenience:

(1) The distinguished symbol is S1.

(2) There is another symbol COMP, distinct from S1, which is expanded only by the rule COMP → SENTB S1 SENTB.

(3) The sentence boundary symbol SENTB is a terminal symbol. Neither it nor S1 appears in the right member of any rule except the one given above in rule 2.

---

* For a discussion of the case where the base component is a context sensitive grammar see Section 3.7 of reference [2].

Our input format for the base component represents the CF rule $A_1 \rightarrow A_2 \ldots A_n$ by the list $(A_1\ A_2 \ldots A_n)$. We do not make use of a notation that allows CF rules with optional or disjunctive sets of constitutents, which are usually denoted by parentheses and brackets respectively. Were we to allow such notation, however, we would not simply produce the expanded CF grammar that this notation denotes. Instead, we would obtain an equivalent grammar using a generalization of the algorithms found in sectinn 5 of reference [3].

For example, the complex CF rule

$$(i)\quad A_1 \rightarrow (A_2)\left\{\begin{matrix}A_3\\A_4\\A_5\end{matrix}\right\} A_6 \left\{\begin{matrix}A_7\left(\left\{\begin{matrix}A_8\\A_9\end{matrix}\right\}\right)\left(\left\{\begin{matrix}A_3\\A_4\\A_5\end{matrix}\right\}\right)(A_{10})\\ A_{11}\ (A_{12})\end{matrix}\right\}$$

denotes 300 separate CF rules. If, however, in place of this rule we substitute the set of rules (ii) below, we generate the same language.

(ii) $A_1 \rightarrow A_2 B_2 A_6 B_3$

$A_1 \rightarrow B_2 A_6 B_3$

$B_2 \rightarrow A_3$

$B_2 \rightarrow A_4$

$B_2 \rightarrow A_5$

$B_3 \rightarrow A_7\ B_4\ B_5$

$B_3 \rightarrow A_7 B_5$

$B_3 \rightarrow A_7$

$B_3 \rightarrow A_{11} A_{12}$

$B_3 \rightarrow A_{11}$

$B_4 \rightarrow A_8$

$B_4 \rightarrow A_9$

$B_5 \rightarrow B_2 A_{10}$

$B_5 \rightarrow B_2$

$B_5 \rightarrow A_{10}$

Furthermore, structural descriptions of sentences with respect to a grammar G containing rule (i) may be obtained from those with respect to a grammar G′ in which rules (ii) are substituted for rule (i) by merely drawing a line through all nodes B_i which are nonterminal symbols of G′ but not G. It is possible to mechanically produce such a grammar G′ that is strongly equivalent in this sense to a CF grammar G that is written in parenthesis and brace notation. Not only is this more economical of storage space than merely expanding G to simple CF rules, but the analysis of sentences with respect to G′ is, at least for many CF analysis procedures, more efficient than is analysis with respect to the expanded form of G.

Although we could have programmed the mechanical derivation of equivalent grammars, we did not do so because the base component is not frequently modified during the development of a transformational grammar, and it is not difficult to produce a particular equivalent grammar by hand. This was,

for example, done for a CF grammar related to the MITRE Junior Grammar [4], reducing the number of CF rules required from over 5000 rules to just 305 rules.

The extension of the base component to make use of complex features in the lexical rewriting rules is currently being programmed. However, for the version of the program that is presently available lexical rewriting must be accomplished by CF rules.

The transformational component consists of an ordered set of transformations, each of which maps a tree structure into another tree structure. These transformations are applied in order and in a cyclic fashion until they are no longer applicable, the output of a transformation at each stage serving as the input to the next transformation. The first transformation is applied initially to a "deep structure" given by the base component, and the ultimate structures produced are called surface structures. For details on the transformational cycle see reference [1] or [2]. The reader is also referred to those sources for definitions of such terms as structural index and proper analysis, to which we refer in the sequel.

We distinguish two types of transformations, singulary and binary, although we need not have formally made the distinction. Singulary transformations must precede binary transformations in the ordering sequence. Binary transformations are distinguished by the fact that they always refer to two levels of embedding, and they always collapse those two levels of embedding to a single embedding level. (The nth embedding level is defined as that portion of a tree dominated by precisely n occurrences of a symbol COMP whose immediate constituents are SENTB S1 SENTB.) The structural index of every binary transformation must contain two occurrences of the sentence boundary symbol SENTB, and the structural change must indicate that both occurrences are deleted. The nodes which satisfy the terms of the structural index of some singulary transformation must all be contained in some currently deepest embedding level, so of course no singulary transformation can have SENTB as a structural index term nor can any rule schema term X of a singulary transformation be satisfied by a sequence of trees containing the symbol SENTB.

Binary transformations are repeated at one upper embedding level until no longer applicable. Singulary transformations may also be designated as REPEATED in their application at a given

point in a derivation. Singulary transformations may also be designated as optional (OPT) or obligatory (OBLIG) with respect to their performance if applicable. In the present version of our analysis program we assume all transformations are optional, reserving for a synthesis phase the elimination of derivations that violate the requirements of some obligatory transformation. Inclusion of optional-obligatory considerations into the analysis procedure itself is discussed in [2] but its programming has not been completed.

The structural change specifies the changes to be performed on the tree being transformed in terms of two elementary transformations, sister-adjunction and substitution (which includes deletion). Sister adjunction is an operation that adjoins two or more trees in an indicated order and substitution is an operation which substitutes an adjoined sequence of trees or a single tree or a null tree for a given subtree R′ of some tree R. If an adjoined set of trees or a single tree is substituted for R then each substituted tree is connected to that node of R which immediately dominated R′. If, however, a null tree is substituted for R′ then not only is R′ deleted, but also higher structure emanating from R′ up to the first node that dominates two or more nodes.

The structural change is an n-tuple where n is the number of terms in the corresponding structural index. Each of its n terms consists of either (1) the integer zero, denoting substitution of the null tree, (2) an integer j between 1 and n, denoting substitution of the jth tree of the proper analysis which satisfies the structural index, (3) a specific morpheme, which need not necessarily be a terminal symbol of the base component, or (4) the adjunction of two or more of the members of (2) and (3). The ith term of the structural change denotes the structure which is to be substituted for the ith tree of the proper analysis in question.

If the jth term of the structural index is a rule schema symbol X, then the jth term of the structural change must be simply the number j, denoting that the structure designated by X is not to be modified.

In specifying the domain of a transformation it is necessary not only to make use of the notion of a proper analysis with respect to a structural index but also to utilize supplementary conditions. Many different types of conditions have been suggested by linguists as appropriate for natural languages, and a few types of conditions appear to be universally employed.

by transformational grammarians. One such type is the condition that two trees satisfying different terms of a structural index be identical, and another type is the condition that a tree be dominated by a prescribed node and/or dominate other prescribed nodes. Changes are currently being made to permit use of all the different types of conditions that Rosenbaum uses in his core grammar [5]. The currently available version of the program, however, allows only the requirement of equality and the requirement that a node dominate a prescribed sequence of nodes.

To make clearer our formalism for transformations we present an example using the LISP S-expression notation which we use as our input format. Consider the transformation

```
(OBLIG (((PRE)) NP AUX V NP BY PASS ((ADV)))
       (1 5 (3 BE EN) 4 0 6 2 8)
       ( ) PASSIVE).
```

This transformation is an ordered 5-tuple expressed as a list of five elements. The first element, OBLIG in our example, denotes that this transformation is an obligatory singulary transformation. Other choices for this first element are OPT for optional singulary transformation,

REPEATED for repeated singulary transformation, and BINARY for binary transformation.

The second element of the transformation is its structural index. In our example this is (((PRE)) NP AUX V NP BY PASS ((ADV))). Terms consisting of a list of symbols enclosed in double parentheses denote either no symbol or any one of the symbols thus enclosed. Hence, by (A B (((C)) D) we mean (A B C D) or (A B D) and by (A B ((C D)) E) we mean (A B C E), (A B D E), or (A B E). A term formed by enclosing a list of symbols in a single pair of parentheses denotes the choice of exactly one symbol in all of the possible ways.

The third element is the structural change. Adjunction is indicated by grouping the set of trees to be adjoined, using a pair of parentheses. In our example the optional tree dominated by PRE is to be left alone (substituted for itself) as are the trees dominated by V and BY. The second tree dominated by NP is to be substituted for the first, the morphemes BE and EN are to be adjoined as right sisters to the tree dominated by AUX, the second NP-tree is to be deleted, the first NP-tree is to be substituted for the structure dominated by PASS, and the tree dominated by ADV, if present, is to be left alone.

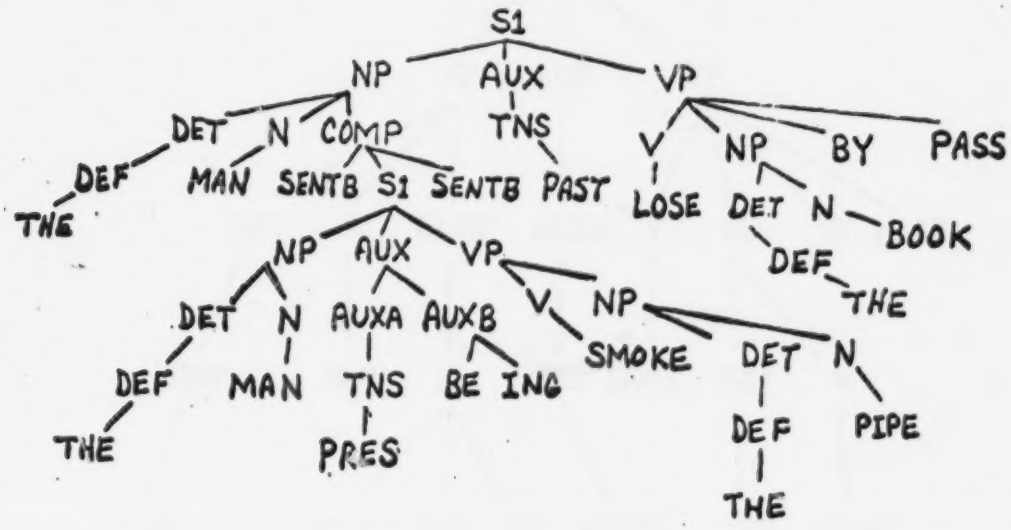
The fourth element is a list of pairs of trees that must be identical if the transformation is to be applicable. For example, identity of the DET-trees and N-trees referenced in the binary transformation

(BINARY (X DET N SENTB DET ((ADJ)) N ((LOC)) AUX VP SENTB X)
(1 (2 6) (3 8) 0 0 0 0 0 (THAT 9) 10 0 12)
((2 5) (3 7)) SUREL)

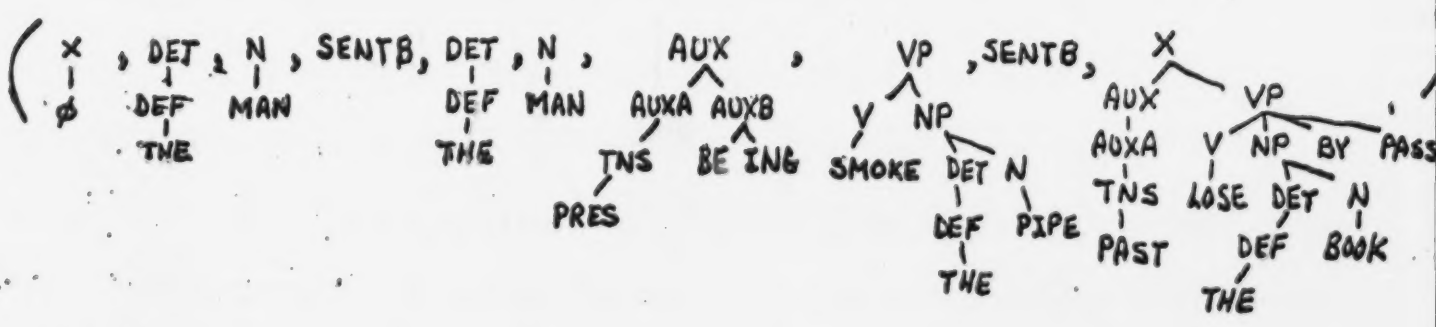
is indicated by the term ((2 5) (3 7)). If a transformation requires no conditions of identity be met, it is still necessary to indicate this by making the fourth element of the transformation ( ) or NIL.

The fifth element of a transformation is simply the name of the transformation. A sixth element can optionally be inserted (making the transformation a 6-tuple) to indicate that a node of a tree satisfying a given structural index term must dominate a prescribed sequence of nodes. For example, if in the previous BINARY transformation we add the sixth element ((2 (DEF))(10 (V NP))) then we indicate that the first DET-tree of a proper analysis must dominate the node DEF and the VP-tree must dominate V NP. It should be noted that we do not require the dominated nodes to constitute a complete proper analysis of the node that dominates them.

To illustrate the application of transformations to a deep structure given by a base component let us consider a transformational component consisting of the singulary transformation PASSIVE and the BINARY transformation previously given, the former ordered first as we require. We apply these rules to the base tree

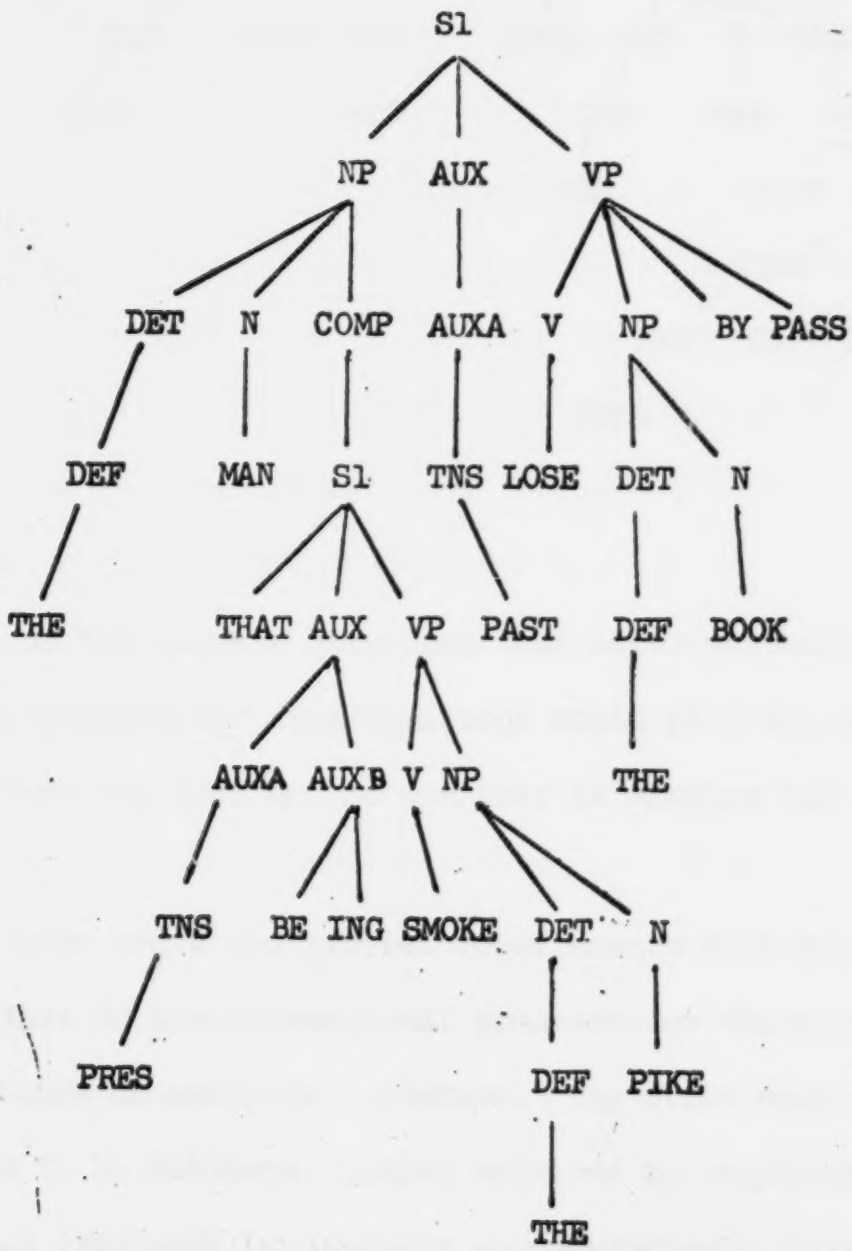


On the first cycle of applying rules the singulary transformation is not applicable but the binary transformation is because the proper analysis

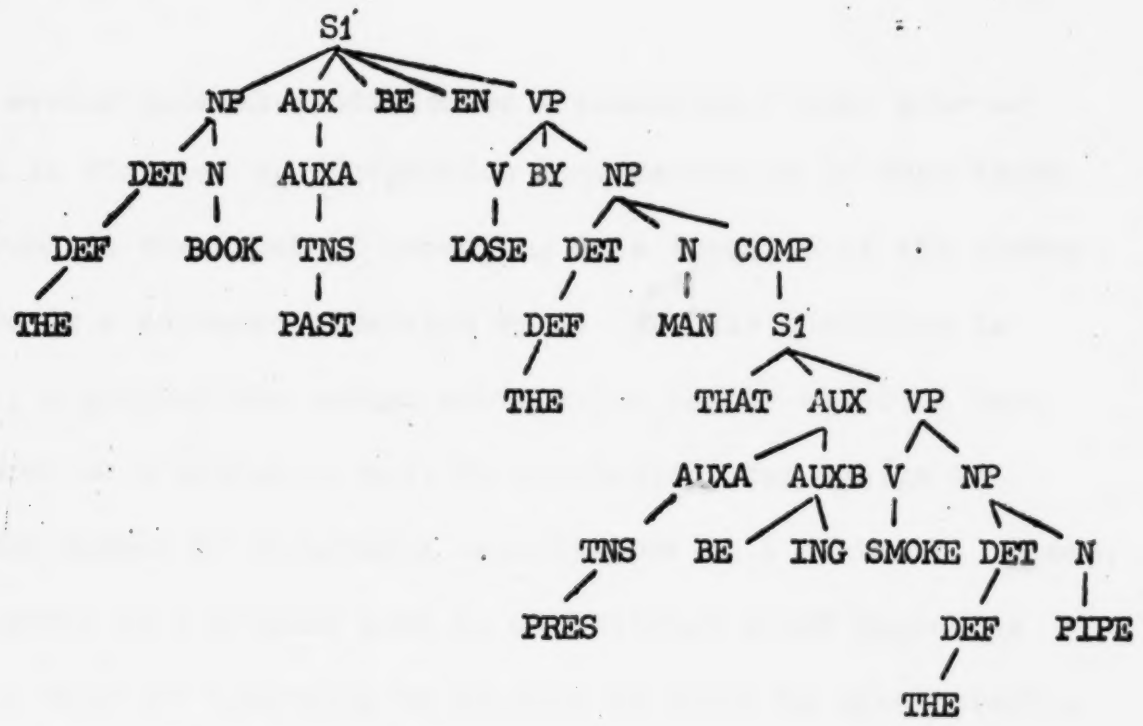


satisfies the structural index of the binary transformation.

Performing this transformation gives the tree


S1
NP AUX VP
DET N COMP AUXA V NP BY PASS
DEF MAN S1 TNS LOSE DET N
THE THAT AUX VP PAST DEF BOOK
AUXA AUXB V NP THE
TNS BE ING SMOKE DET N
PRES DEF PIKE
THE

Going through the transformational cycle again from the beginning, we see that the PASSIVE transformation is applicable to this tree. Performing it gives


S1
NP AUX BE EN VP
DET N AUXA V BY NP
DEF BOOK TNS LOSE DET N COMP
THE PAST DEF MAN S1
THE THAT AUX VP
AUXA AUXB V NP
TNS BE ING SMOKE DET N
PRES DEF PIPE
THE

which is the surface structure that after verb-affix permutation and morphophonemic tranformations would give the sentence, "The book was lost by the man that is smoking the pipe."

There are a few general requirements that must be made on the class of transformational grammars for which our program consitutes an analysis procedure. The first such general condition, due to G. H. Matthews, limits deletion by requiring that each deleted tree must be uniquely reconstructable from the transformed tree. More precisely, the number of every term of the structural index which is not a terminal symbol or transformationally introduced

morpheme must either appear somewhere in the corresponding structural change or else must be equated to some number that does occur in the structural change.

A second general condition on a transformational grammar G which is dictated by recognition considerations is that there be a bound on the depth of embedding as a function of the number of words in a sentence generated by G. If this condition is not met, a grammar can assign arbitrarily deeply embedded base structures to a sentence, and, in particular, can assign an unbounded number of structural descriptions to a sentence. Hence, requirements on a grammar must be established which guarantee that the depth of embedding be bounded in order to make possible a recognition procedure whose computation is bounded as a function of sentence length. There are several ways of establishing such requirements. These are discussed in reference [2] but are not particularly germane to the user of the existing LISP analysis program, who must fix a maximum depth of embedding prior to analyzing a particular sentence (or set of sentences). Ensuing analysis will be valid only for sentences whose maximum depth of embedding does not exceed this limit.

A similar requirement on a transformational grammar is that there be a bounded number of occurrences of the symbol COMP at

any embedding level. This requirement is always met in grammars of practical linguistic interest where recursion is permitted only through use of the symbol COMP.

A final requirement is that the number of consecutive applications of a repeated transformation which are allowed in a single cycle must be bounded. As with the previous requirement no loss of practical descriptive facility is caused by bounding the number of consecutive applications of some repeated transformation, for example, to the number of terminal symbols in the sentence being analyzed.

This concludes our rather informal description of the class of grammars with respect to which our program finds all structural descriptions of a given sentence. In the following section we give operating instructions and input-output formats. It is hoped that the examples given will illustrate the class of grammars for which our computer program constitutes a general analysis procedure.

Operating Instructions

We limit our consideration of computer operating instructions and input-output formats to the IBM 7044 - 7094 LISP 1.5 computer implementations of our analysis algorithm. The versions of the program for the MIT CTSS system and the AFCRL UNIVAC M-460 computer are essentially the same.

Starting with a basic LISP 1.5 system a packet of cards is read, and overlord instructions which define and compile all necessary functions are executed. Included in this packet are cards which set the values of certain constants which are referenced as free variables by the various functions.

To use the system it is first necessary to "set-up" (i,e. define) both the base component and the transformational component of a grammar. A list of base component rules must be set as the value of the constant named GRAMMAR. For example, for the grammar

S1 → NP VP1

VP1 → V1

V1 → APPEARS

NP → IT COMP

S1 → NP VP2

S1 → NP VP2 MAN

VP2 → V2 NP

NP → DET N

N → DOGS

N → CATS

DET → THE

MAN → BY P

V2 → LIKE

COMP → SENTB S1 SENTB

we evaluate

CSET (GRAMMAR ((S1 NP VP1)(VP1 V1)(V1 APPEARS)(NP IT COMP)
(S1 NP VP2)(S1 NP VP2 MAN)(VP2 V2 NP)(NP DET N)(N DOGS)
(N CATS)(DET THE)(MAN BY P)(V2 LIKE)))

Note that the rule COMP → SENTB S1 SENTB is understood and should not be included as an element of the list GRAMMAR.

In addition to the list GRAMMAR two other constant lists must be set up in specifying the base component. One is the list TERMTABLE which associates with each nonterminal symbol of the base component a unique terminal symbol. We illustrate the format for the CF grammar just given.

```
CSET (TERMTABLE ((S1 S10)(VP1 VP10)(V1 V10)(NP NP0)(VP2 VP20)
     (N N0)(DET DET0)(MAN MAN0)(V2 V20)(COMP COMP0)(X X0)))
```

This particular fabrication of terminal symbols (i.e, adding 0 to the corresponding nonterminal) is not essential. Any LISP atoms not already used can be used. Note that the rule X → X0 must be included. The system requires this in order not to classify the rule schema symbol X as a terminal symbol of the base component. The necessity to fix the value of TERMTABLE stems from a requirement we have of performing CF analysis on a sequence of trees, the nodes which dominate these trees serving as terminal symbols for this analysis. We could have programmed the computation of a suitable value of TERMTABLE from the list GRAMMAR, but we never got around to doing so.

The last list which must be defined in conjunction with the base component is the set of rules reflecting derived constituent structure, that is, phrase structure distinct from that implied by just the base component but which can, nevertheless, be produced by the transformational component operating on some base structure. It is shown in [2] that there may be an unbounded number of such derived constituent structure rules implied by a transformational grammar.* It is further shown, however, that there are a bounded number of such rules that can arise from a deep structure whose depth of embedding is bounded. An algorithm is given for

* Friedman [11] shows that even if schemata are permitted there can be no algorithm for constructing a surface grammar for an arbitrary transformational grammar.

finding a set of phrase structure rules which includes all of the derived constituent structure rules that could be produced by a given transformational grammar operating on a structure whose maximum depth of embedding is less than a given integer. Conditions on a transformational grammar are given in [2] which bound the maximum depth of embedding as a function of sentence length. Hence, for a sentence of length n it is possible to find a set of phrase structure rules that includes all the derived constituent structure rules produced by a derivation of that sentence. As our program is presently organized, however, it is necessary to decide upon a maximum depth of embedding for which a subsequent analysis will be valid and to provide the sufficient set of derived constituent structure (dcs) rules. A list of these rules must be set to be the value of the constant AUXRULES.

A program to mechanically compute a sufficient set of dcs rules for a given maximum depth of embedding has not been completed. Hence, at present two choices are possible. One can either produce by hand computation a sufficient set of dcs rules, using the method of [2], or else one can use for a particular sentence just a set of rules that is known to include all the dcs rules implied by some derivation(s) of that sentence. This latter procedure, of course, does not constitute a generally valid analysis procedure. It can fail to find certain ambiguous structural descriptions,

and it requires at least a mental generation of the sentence. It is, however, worthwhile to use this way of reducing the number of rules in AUXRULES just to insure that a few sentences are indeed generated as intended by a relatively economical computation.

For the small CF grammar given above and the set of transformations to follow we will subsequently make use of the rules:

```
CSET (AUXRULES ((COMP S1)(S1 THAT NP VP2)
     (NP IT)(VP2 BE ED V2)(MAN BY NP)
     (S1 THAT NP VP2 MAN)(S1 NP VP1 S1)
     (S1 FOR TO NP VP2)(S1 FOR NP VP2)
     (VP2 TO V2 NP)(S1 FOR VP2)(S1 VP2)
     (S1 VP2 MAN)(S1 FOR TO NP VP2 MAN)
     (S1 FOR NP VP2 MAN)(VP2 TO BE ED V2)
     (S1 FOR VP2 MAN)))
```

When the three lists GRAMMAR, TERMTABLE, and AUXRULES have been set, it is necessary to execute the LISP function of no arguments SETCFG. The command SETCFG ( ) sets up several tables necessary for subsequent CF grammar analysis. The CF analysis procedure used is the so-called SBT algorithm of reference [3].

After setting up the base component it is possible to perform analysis of sentences with respect to the CF grammar thus defined. (If only analysis with respect to a given CF

grammar is wanted, set its rules as the value of GRAMMAR and set both TERMTABLE and AUXRULES to NIL.) The function used is CFREC. Its first argument is a sentence to be parsed, and its second argument is the nonterminal symbol which is to be considered the distinguished symbol of the grammar for the parsing.

For the above grammar we have

CFREC ((THE DOGS LIKE THE CATS) S1) =

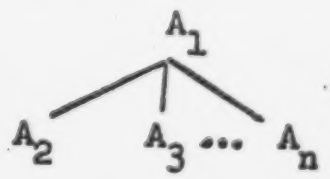
(((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS))))NIL))

and

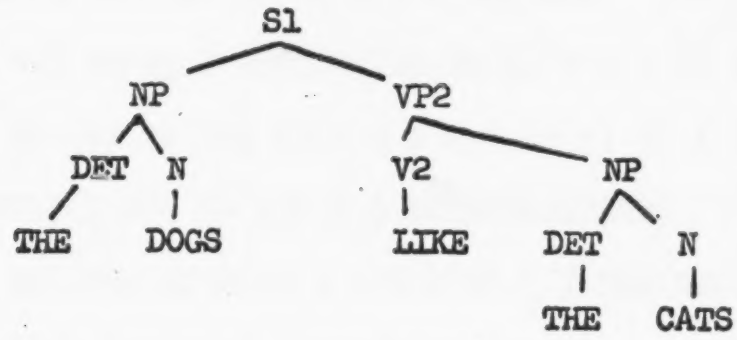
CFREC ((THE DOGS) NP) =

(((NP(DET THE)(N DOGS))NIL))

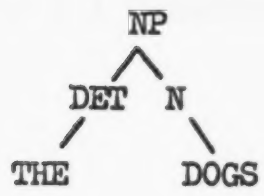
The two values of CFREC above illustrate our notation for representing trees. The outermost parentheses reflect the fact that the output is a list of structural descriptions in general (although in our examples each of the two strings has one parsing). Each parsing is expressed as a list of two elements. The first element is a representation of a tree which is a structural description of some initial substring of the first argument of CFREC and the second element is the list of remaining unused terminal symbols of the first argument which are not included in the substring parsed. The tree representation simply denotes the structure



by the linear notation $(A_1\ A_2\ A_3 \ldots A_n)$. The two trees represented above are:



and



The transformational component is set up by the function SETTRANS whose single argument is a list of ordered transformations in the format already discussed. This function computes a list of so-called inverse transformations and fixes the value of the constant TRANS as this list. The inverse transformations and their role in the analysis procedure are discussed in [2]. The format of a set of transformations for the base component already given is:

```
SETTRANS ((
(OPT (X IT NP X)(1 2 (FOR TO 3) 4) ( ) C/INTRO1)
(OPT (X IT NP X)(1 2 (THAT 3) 4) ( ) C/INTRO2)
(OPT (X NP (V1 V2) NP BY P X)(1 4 (BE ED 3) 0 5 2 7) ( ) PASSIVE)
(OPT (X FOR TO NP (V1 V2 BE) X) ( 1 2 0 4 (3 5) 6) ( ) C/PLACEMENT)
(OPT (X IT S1 VP1)(1 2 0 (4 3)) ( ) EXTRAPOSITION)
(OPT (X IT (V1 V2) FOR NP X)(1 5 3 4 0 6) ( ) PRO/REPLACE)
(OPT (X FOR TO X)(1 0 3 4) ( ) FOR/DELETION)
(BINARY (IT SENTB NP VP2 ((MAN)) SENTB VP1)
        (1 0 3 4 5 0 7) ( ) CHANGELEVELS)))
```

We give below the list of inverse transformations which is set to be the value of the constant TRANS when the above function is executed. We will refer to these inverse transformations subsequently in discussing several analysis examples.

(((X TO X)(1 FOR 2 3) OPT FOR/DELETION(TO))

((X NP(V1 V2) FOR X)(1 IT 3 4 2 5) OPT PRO/REPLACE(FOR))

((X IT VP1 S1)(1 2 4 3) OPT EXTRAPOSITION(IT))

((X FOR NP TO(V1 V2 BE)X)(1 2 4 3 5 6) OPT C/PLACEMENT(FOR TO))

((X NP BE ED(V1 V2) BY NP X)(1 7 5 2 6 P 8) OPT PASSIVE(BE ED BY))

((X IT THAT NP X)(1 2 4 5) OPT C/INTRO2(IT THAT))

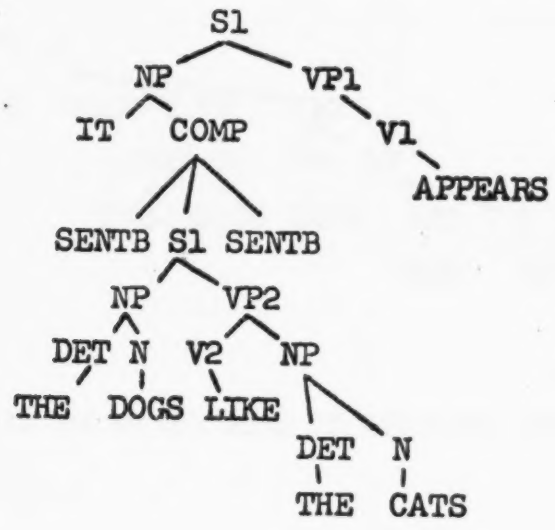
((X IT FOR TO NP X)(1 2 5 6) OPT C/INTRO1(IT FOR TO))

(BINARY(IT NP VP2((MAN))VP1)(1 COMP 5)(2 3 4) CHANGELEVELS(IT)))

Inverse singulary transformations are represented as five element lists. The first element is the inverse structural index. The second element is the inverse structural change. The third element denotes the type of singulary transformation in question, and the fourth element is the name of that transformation. The fifth element is a sequential list of the morphemes appearing in the inverse structural index. Aside from a different ordering, inverse binary transformations differ only in that in place of an inverse structural change there are two elements, the third and fourth, giving structural changes corresponding to the matrix and constituent sentences, respectively. We will defer further discussion of inverse transformations to the next section.

To illustrate the generative application of transformational

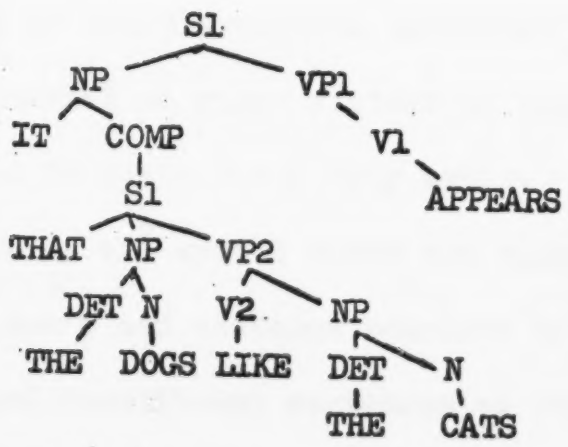
rules for the simple grammar given above we give a derivation of the sentence, "It appears that the dogs like the cats." The underlying base structure is



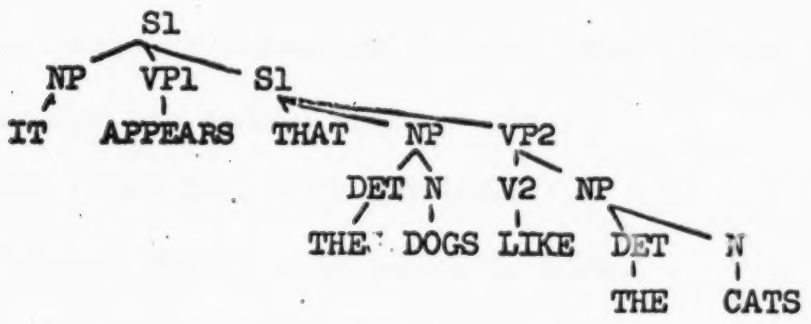
On the first cycle no singulary transformations apply. There is, however, a proper analysis of the base tree with respect to the structural index of the binary transformation,

(IT, SENTB, NP, VP2, ((MAN)), SENTB, VP1).

Applying this transformation merely erases the sentence boundary symbols SENTB. On the second cyclic application of the transformational rules the transformation C/INTRO2 is applied to give:



Performing the next transformation that applies to this structure, EXTRAPOSITION, gives:



the terminal symbols of which are IT APPEARS THAT THE DOGS LIKE THE CATS.

Analysis of a sentence with respect to a grammar already set up is accomplished by means of the function RECOGNIZE. To analyze the sentence whose derivation appears above, for example, we evaluate RECOGNIZE ((IT APPEARS THAT THE DOGS LIKE THE CATS)). The resulting value is

((((S1(NP IT (COMP ((S1 (NP (DET THE)
(N DOGS))(VP2 (V2 LIKE)(NP (DET THE)
(N CATS)))))CHANGELEVELS))(VP1
(V1 APPEARS))))(C/INTRO2 EXTRAPOSITION)))

Note that the lists of transformations performed at each level of embedding are attached as right sisters to the sets of structures dominated by S1 to which they apply. Note also that all occurrences of the symbol SENTB are missing in this output. We simply don't add sentence boundary symbols when connecting matrix and constituent sentences as reflected by the rule COMP→SENTB S1 SENTB. This could, of course, be

easily done, but the formalization of this paper was not originally programmed, and later it wasn't felt important enough to bother changing the output to conform with the sentence boundary formalism decided upon. Other analysis examples for the sample grammar are

```
RECOGNIZE ((THE DOGS LIKE THE CATS)) =
((((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)
   (NP(DET THE)(N CATS)))))NIL))

RECOGNIZE ((THE DOGS APPEARS TO LIKE THE CATS)) =
((((S1(NP IT(COMP((S1(NP (DET THE)
   (N DOGS))(VP2(V2 LIKE)(NP (DET THE)
   (N CATS)))))CHANGELEVELS))(VP1(V1 APPEARS))))
   (C/INTRO1 C/PLACEMENT EXTRAPOSITION
    PRO/REPLACE FOR/DELETION)))

RECOGNIZE ((IT APPEARS THAT THE CATS BE ED LIKE BY THE DOGS)) =
((((S1(NP IT(COMP((S1(NP(DET THE)
   (N DOGS))(VP2(V2 LIKE)(NP(DET THE)
   (N CATS)))(MAN BY P)))CHANGELEVELS))
   (VP1(V1 APPEARS))))(C/INTRO2 PASSIVE
    EXTRAPOSITION))(((S1(NP IT(COMP
  ((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)
   (NP(DET THE)(N CATS)))(MAN BY P)))
    PASSIVE CHANGELEVELS)) (VP1(V1 APPEARS))))
  (C/INTRO2 EXTRAPOSITION)))
```

```
RECOGNIZE ((THE CATS APPEARS TO BE ED LIKE BY THE DOGS))

((((S1(NP IT(COMP((S1(NP(DET THE)
  (N DOGS))(VP2(V2 LIKE)(NP(DET THE)
  (N CATS)))(MAN BY P)))CHANGELEVELS))
  (VP1(V1 APPEARS))))(C/INTRO1 PASSIVE
   C/PLACEMENT EXTRAPOSITION PRO/REPLACE
   FOR/DELETION))(((S1(NP IT(COMP
 ((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)
  (NP(DET THE)(N CATS)))(MAN BY P)))
   PASSIVE CHANGELEVELS))(VP1(V1 APPEARS))))
  (C/INTRO1 C/PLACEMENT EXTRAPOSITION
   PRO/REPLACE FOR/DELETION)))
```

To simplify this sample grammar we have omitted a verb-affix permutation transformation, number agreement transformations, and morphophonemic transformations. This simple grammar was written by Peter Rosenbaum as an exercise in learning the conventions required by our system.

We observe that the last two sentences can each be produced by two distinct derivations which differ only with respect to whether the PASSIVE transformation is performed on the first cycle or on the second. If this transformation is made obligatory, only a single derivation is allowed, of course.

If the user wishes to follow the course of the recognition procedure, an option is possible that gives extra printing. This is indicated by evaluating CSET (SW3 T). Conversely

setting the value of the constant SW3 to NIL causes the extra printing to be repressed. If SW3 is set to T, the following printing results for a simple example:

```
RECOGNIZE((THE DOGS LIKE THE CATS))
CONTINUATION NO. 0
(THE DOGS LIKE THE CATS)
FOR/DELETION
PRO/REPLACE
EXTRAPOSITION
C/PLACEMENT
PASSIVE
C/INTRO2
C/INTRO1
CHANGELEVELS
BASE
((((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS)))))NIL))
```

Names of transformations are printed as the corresponding inverse transformations are tested for applicability. In the above simple example no inverse transformations were applicable, so after exhausting them the given sentence was parsed with respect to the base component CF grammar, giving the required structural description.

A more complicated example is the following:

```
RECOGNIZE((IT APPEARS THAT THE DOGS LIKE THE CATS))
CONTINUATION NO. 0
(IT APPEARS THAT THE DOGS LIKE THE CATS)
FOR/DELETION
PRO/REPLACE
EXTRAPOSITION
((X) IT (VP1(V1 APPEARS))(S1 THAT(NP(DET THE)(N DOGS))(VP2
 (V2 LIKE)(NP(DET THE)(N CATS)))))
(IT THAT THE DOGS LIKE THE CATS(VP1(V1 APPEARS)))
PSTEST
Passed
***1TH CONTINUATION

C/PLACEMENT ..
PASSIVE
C/INTRO2
C/INTRO1
CHANGELEVELS
BASE
     NO CONTINUATION

CONTINUATION NO. 1
(IT THAT THE DOGS LIKE THE CATS(VP1(V1 APPEARS)))
C/PLACEMENT
PASSIVE
C/INTRO2
((X) IT THAT (NP(DET THE)(N DOGS))(X LIKE THE CATS(VP1(V1 APPEARS))))
(IT(NP(DET THE)(N DOGS)) LIKE THE CATS (VP1(V1 APPEARS)))
PSTEST
```

```
Passed
***2TH CONTINUATION

C/INTRO1
CHANGELEVELS
BASE
     NO CONTINUATION

CONTINUATION NO. 2
(IT(NP(DET THE)(N DOGS)) LIKE THE CATS(VP1(V1 APPEARS)))
C/INTRO1
CHANGELEVELS
(IT(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS)))
 EMPTY (VP1(V1 APPEARS)))
***3TH CONTINUATION

(IT COMP(VP1(V1 APPEARS)))
PSTEST
Passed
((NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS))))
PSTEST
Passed
CONTINUATION NO. 4

((NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS))))
FOR/DELETION
PRO/REPLACE
EXTRAPOSITION
C/PLACEMENT
PASSIVE
```

```
C/INTRO2
C/INTRO1
CHANGELEVELS
BASE
((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS)))))
***5TH  CONTINUATION

CONTINUATION NO. 5
(IT(COMP((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)
 (N CATS)))))CHANGELEVELS)(VP1(V1 APPEARS)))
BASE
((S1(NP IT(COMP((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP
 (DET THE)(N CATS)))))CHANGELEVELS))(VP1(V1 APPEARS))))

CONTINUATION NO. 3
(IT(NP(DET THE)(N DOGS)) LIKE THE CATS(VP1(V1 APPEARS)))
BASE
NO CONTINUATION
((((S1(NP IT(COMP((S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP
 (DET THE)(N CATS)))))CHANGELEVELS))(VP1(V1 APPEARS))))
 (C/INTRO2 EXTRAPOSITION)))
```

We give a brief commentary on this print out below. It is not meant to be an explanation of how the analysis procedure works. Rather it merely illustrates the output format for displaying the intermediate structure that is pieced together at each stage of the analysis in terms that should be clear to the reader familiar with reference [2]. Other readers

will understand what sequence of computations is being made but will not have any understanding of why this sequence constitutes a complete analysis.

The FOR/DELETION and PRO/REPLACE inverse transformations are not applicable. The given sentence is analyzable into a sequence of trees satisfying the inverse structural index of the inverse EXTRAPOSITION transformation, however. This sequence of trees is given on the line after the name of the transformation EXTRAPOSITION, and on the subsequent line is found the transformed sequence of trees formed by the inverse EXTRAPOSITION transformation. The PSTEST appearing on the next line refers to a necessary test which must be passed if a sequence of trees resulting from an inverse transformation is to lead to a valid structural description. If such a sequence of trees is to be further considered, it must be a proper analysis of a derived structure tree at some point in the derivation of the sentence in question. Hence, it must be possible to build up structure from this sequence of trees to the sentence symbol S1, using both the original base rules and the derived constituent structure rules included in the list AUXRULES. The message PASSED on the line after PSTEST indicates that this necessary test was passed.

The next line ***1TH CONTINUATION indicates that the transformed sequence of trees is being stored on a pushdown list as continuation number one.

The subsequent list of transformation names refer to inverse

transformations considered for potential performance on the untransformed sequence (IT APPEARS THAT THE DOGS LIKE THE CATS). When this continuation eventually fails, the transformed sequence of trees (continuation number one) is taken from the pushdown list.

The inverse transformations C/PLACEMENT and PASSIVE do not apply but C/INTRO2 does apply. The test PSTEST on the transformed sequence of trees is passed and this sequence is stored as continuation number two. The continuation resulting from not applying the inverse C/INTRO2 transformation fails with no further inverse transformations being applicable and continuation two is next considered. Inverse C/INTRO1 is not applicable, but the inverse of binary transformation CHANGELEVELS is applicable. In the case of inverse binary transformations the order of considering transformed and untransformed sequences is reversed. The untransformed sequence is stored away as continuation three, and that part of the inverse binary transformation is performed which gives the matrix sentence continuation. This sequence,

(IT COMP(VP1(V1 APPEARS))),

passes the test PSTEST as does the constituent sentence sequence ((NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS)))).

The constituent sentence continuation (which is labelled continuation number four) is next subjected to the entire inverse transformational cycle. No transformations are performed, but

the base structure

(S1(NP(DET THE)(N DOGS))(VP2(V2 LIKE)(NP(DET THE)(N CATS))))

is found. This structure is then attached to the symbol COMP of the matrix sentence continuation and stored as continuation number five. This continuation is immediately retrieved and built up to S1 by means of the base component rules, giving a complete structural description. Continuation three is then pursued. This continuation dies immediately because it is not possible to build the sequence up to S1 using base component rules. Finally, the structural description found is printed out.

In addition to the printing option controlled by the value of the constant SW3, there are several other constants whose value gives options. Constants NOTEST and NOTEST1 determine whether the test PSTEST is to be carried out after performance of singulary and binary transformations, respectively. A value of NIL indicates the tests are to be performed and a value of T indicates they are not to be performed. Setting ONECPL to T is appropriate only when it is known prior to analyzing a sentence that that sentence contains no more than a single COMP at any level of embedding. If ONECPL is NIL no such simplifying assumption is made. SDSASFOUND is a constant which, if its value is T, signals that each structural description is to be printed as soon as it is found. Setting SDSASFOUND to NIL indicates structural descriptions are to be printed only after the analysis procedure has been completed. Finally, the

value of NOOFPARSINGS controls the number of structural descriptions with respect to a given set up context-free grammar which are to be computed by the function CFREC. The self-explanatory possible values are ONE and ALL, and only the latter is appropriate for use in analysis with respect to a transformational grammar. Setting NOOFPARSINGS to ONE is useful if for some reason only a single structural description of a sentence with respect to a given CF grammar is required.

Decks of cards containing the version of the system we have described are available to any interested group. These are appropriate for the IBM 7094 and 7044 LISP systems. They consist of LISP packets which define and compile the functions needed, set required values of all constants, set up the simple grammar of this paper, and analyze a few sentences. This, in conjunction with the present paper, should prove sufficient to permit the testing of transformational grammars written by individual writers.

Discussion

We have defined a class of transformational grammars and have described how to use a computer program which determines all of the structural descriptions of a sentence with respect to an arbitrary member of this class of grammars. It is hoped that we have done this well enough to permit the use of our syntactic analysis system by any interested investigator.

Our work on syntactic analysis contrasts in several respects with that of other groups. The principal difference, obviously, is that few efforts have been directed toward the analysis of sentences with respect to transformational grammars. Limiting our attention to analysis procedures for transformational grammars, we find work by Matthews [6,7,8] Zwicky et al [4,9] and Kuno [10] . Matthews was the first to write on the transformational recognition problem. Briefly, he showed how for every structural description of each sentence of a language defined by some transformational grammar, it is possible to set up a unique integer (specifier) that denotes a unique transformational derivation of the sentence in question. By obtaining a bound f(n) on the specifier of a sentence consisting of n terminal symbols, he demonstrated that it is in principle possible to obtain all structural descriptions of a given sentence by applying the rules through a synthesis procedure no more than a bounded number of ways. He observed that actually following such a procedure would be prohibitively time consuming, and he proposed to make the procedure feasible by performing certain preliminary analyses on the sentence so as to preclude large numbers of potential specifiers and to reduce the amount of exhaustive sentence synthesis required to a reasonable magnitude. In his last paper he gave details of how a system of specifiers could be set up for a so-called deep structure transformational grammar, and he began to consider details of a preliminary analysis procedure. This system has not been

programmed to date. Our analysis procedure finds all structural descriptions of a given sentence along with, possibly, one or more incorrect structural descriptions. For a discussion of why a synthesis phase is necessary see Section 3.8 of [2]. Hence, this procedure must be used as a preliminary analysis procedure in conjunction with an analysis-by-synthesis algorithm as suggested by Matthews. Our procedure differs from his in that we do not eliminate blocks of sequential specifiers, testing other blocks by means of a synthesis component. Instead, we generate a set of potential specifiers that must include every valid specifier, and we test each of these with a subsequent synthesis.

A distinctly different approach has been utilized by the Language Processing Techniques Sub-department of the MITRE Corporation. Briefly, they make use of a so-called CF surface grammar that assigns to each sentence all of the final derived constituent structure trees that are specified by a given transformational grammar. The surface grammar may also assign unwanted, erroneous derived constituent structure or even generate sentences not defined by the given transformational grammar. In the MITRE approach inverse transformations which map trees into trees are applied in reverse generative order to the set of surface trees given by the surface grammar, and the resulting trees are checked to insure that they satisfy the given base component phrase structure rules. Finally, potential structural descriptions thus obtained are submitted to a synthesis component for verification or rejection.

The work by Zwicky et al at the MITRE Corporation has been directed toward achieving somewhat different goals than our own. Whereas we insisted upon providing an analysis procedure that is valid for a class of grammars, the work at MITRE has been concerned with a single grammar. (Although the scope of this grammar has been continually extended, at every point its associated analysis algorithm has been individually constructed for the existing grammar). In addition, we have required that our procedure find all structural descriptions assigned to a sentence by a given grammar. The MITRE analysis program, on the other hand, is not based upon underlying theoretical considerations which guarantee all structural descriptions of a sentence will be found. There are, however, no known instances where the analysis program is known to fail to produce all of the structural descriptions assigned to a sentence by the so-called JUNIOR Grammar of reference [9].

As might be expected, a rather high price is paid for the generality and theoretical validity of our program. This is bourne out by comparative computing times, even though there were tremendous differences in the computers that were used, in the grammars that were considered, in the programming language systems that were employed, and in the implementation details that were utilized. The total processing time of 5.11 minutes

for 28 sentences cited by MITRE contrasts with our own processing time requirements which have ranged from a few seconds to as much as ten minutes per sentence on the IBM 7094. The undoubtedly greater speed of the MITRE program makes its use more feasible for natural language computer programming applications. It is not, however, suitable for use by a linguist who wished to test his own transformational grammar to ascertain that it assigns structural descriptions as intended.

Kuno [10] has also considered the analysis problem for transformational grammars. He has observed that for many transformational grammars which have been written to describe subsets of English it is possible to predict contiguous portions of the deep structure assigned to a sentence from corresponding distinctive local blocks of surface structure. His proposed approach is to make use of a standard form CF grammar which generates the sentences specified by a given transformational grammar and to develop a procedure for mapping structural descriptions assigned by the standard form grammar directly into those deep structures assigned to the sentence by the transformational grammar. No algorithm has yet been developed for computing the required standard form grammar or for devising a general procedure for mapping its structural descriptions into base structures assigned by a given transformational grammar.

We are still working on extending the version of the program described in this paper. The analysis procedure is being modified to be valid for grammars which make use of complex features for restricting the lexical rewriting portion of the base component. We will also permit transformations to make references to features. Additional facilities will also be added to permit the statement of generalized structural conditions on the applicability of transformations. These modifications should permit consideration of grammars employing essentially the formalism used by Rosenbaum in his core grammar [5].

Several additional programs are being written to make our present system easier to use. The mechanical computation of derived constituent structure rules (AUXRULES) required for analysis of sentences whose depth of embedding does not exceed a given constant is being programmed. A synthesis component is also being programmed. This is required not only to eliminate structures which cannot be transformed to produce the given sentence, but also is required as an independently useful tool in testing a transformational grammar. By means of a synthesis component many gross errors should be detected more economically than would be possible if only an analysis program were used.

Finally, we are continually making changes designed to make our analysis procedure more efficient. Changes being made to

increase programming efficiency include development of selected machine-coded subroutines and increased use of LISP property lists. Changes to the underlying theoretical analysis procedure will probably do even more to increase efficiency. Improvements currently being programmed include modification of the analysis procedure (1) to make continual use of restrictions during the course of an analysis which are associated with the requirement that an obligatory transformation be performed if applicable, and (2) to make use of selectively applicable AUXRULES at different points in an analysis. It is hoped that these changes will result in a system which is significantly more useful than the present system for testing transformational grammars and for application to systems that allow man-machine communication by means of a useful subset of English.

REFERENCES

1. Chomsky, N., (1965). "Aspects of the Theory of Syntax," The M.I.T. Press, Cambridge, Massachusetts.

2. Petrick, S., (1965). "A Recognition Procedure for Transformational Grammars," Ph.D. thesis, M.I.T.

3. Griffiths, T. and Petrick, S., (May,1965). "On the Relative Efficiencies of Context-Free Grammar Recognizers," Comm. ACM., Vol. 8, pp. 289-300.

4. Zwicky, A., Friedman, J., Hall, B., and Walker, D., (1965). "The MITRE Syntactic Analysis Procedure for Transformational Grammars," Proceedings - FJCC, Spartan Books, Washington, D.C.

5. Rosenbaum, P., (March, 1966). "The Core Grammar," Scientific Report, Contract No. AF19(628) - 5127, Air Force Cambridge Research Laboratories, Bedford, Massachusetts.

6. Matthews, G. H., (1961). "The Use of Grammars Within the Mechanical Translation Routine," In H.P. Edmundson (Ed.), Proceedings of the National Symposium on Machine Translation, Englewood Cliffs, N.J., Prentice-Hall.

7. Matthews, G. H., "Analysis by Synthesis of Sentences of Natural Languages," In Proceedings of the 1961 International Congress on Machine Translation of Languages and Applied Language Analysis, Teddington, England, National Physical Laboratory.

8. Matthews, G. H., "Analysis by Synthesis in the Light of Recent Developments in the Theory of Grammar," to appear in Kybernetica, Prague, Czechoslovakia.

9. Zwicky, A., Hall, B., Fraser, B., Geis, M., Mintz, J., Isard, S., and Peters, S., (Dec., 1964). "English Preprocessor Manual, " Information System Language Studies Number Seven, SR - 132, the MITRE Corp., Bedford, Mass.

10. Kuno, S., (May, 1965), "A System for Transformational Analysis," Proceedings of the 1965 International Conference on Computational Linguistics, New York, N. Y.

11. Friedman, J., (Nov., 1965), "An Investigation of Surface Grammars," Paper WP-335, The MITRE Corp., Bedford, Mass.